Australian Government

Department of Defence

Defence Science and
Technology Organisation

Introduction to MATLAB Graphical User Interfaces

Sergey Simakov

Maritime Operations Division
Defence Science and Technology Organisation

DSTO–GD–0442

ABSTRACT

The aim of this course is to develop initial skills for building Graphical User Interfaces (GUIs) in MATLAB7. First we give a summary of MATLAB's graphics object hierarchy and review the syntax for accessing and manipulating object properties. Then we discuss standard user interface components and consider situations when descendants of axes can be used to design purpose-built graphical controls. Programming techniques are analysed using moderately simple conceptual examples and exercises. The structure of application m-files generated by the MATLAB GUI development environment and some techniques for inclusion of Java components and ActiveX controls into MATLAB GUIs are also discussed.

Published by

Maritime Operations Division
DSTO Defence Science and Technology Organisation
PO Box 1500
Edinburgh, South Australia, Australia 5111

Telephone: *(08) 8259 5555*
Facsimile: *(08) 8259 6567*

AR No. 013-447
October, 2005

Introduction to MATLAB Graphical User Interfaces

EXECUTIVE SUMMARY

MATLAB is viewed by many users not only as a high-performance language for technical computing but also as a convenient environment for building graphical user interfaces (GUI). Data visualisation and GUI design in MATLAB are based on the Handle Graphics System in which the objects organised in a Graphics Object Hierarchy can be manipulated by various high and low level commands.

These notes were prepared for a short course on GUI building in MATLAB7. The aim of the course is not to cover all aspects of GUI building, but rather to give a concise exposition of some possible viable GUI design techniques. The MATLAB code discussed in the course examples can also be used as a starting template for developing more complex GUIs.

Each new release of MATLAB brings new objects and categories in the Handle Graphics System. MATLAB7, in which the introduction of many new objects has considerably reshaped the Graphics Object Hierarchy, is also characterised by a significant strengthening of capabilities for using Java and ActiveX.

Though all such changes make the GUI design more flexible and versatile, they also increase the complexity of the Handle Graphics System and require some effort to adapt to.

This course is also an attempt to facilitate such an adaptation through the analysis of moderately simple conceptual examples and applications.

Contents

Figures

Tables

1 MATLAB Graphics Objects

Execution of certain commands in MATLAB is accompanied by the creation of graphical objects. Most of such objects, but not all, can be displayed on the computer screen.

Exercise 1. Consider the results of entering the following commands (the code for Example 1 and the other examples of this section can be found in the folder `Section1`[1]):

```
hf=figure;
ha=axes;
x=[-10:0.1:10];
h=line(x,exp(-x.^2));
```

We can observe that at least 3 graphical objects (of type figure, axes and line) are being created. There is also a graphical object, called *root* which corresponds to the computer screen.

When a graphical object is created, it is assigned a unique number called a handle. The handle of the *root* object is always 0. Graphics handles can be used as input arguments in the commands `get` and `set` to access and manipulate properties of objects.

Exercise 2. Continue the above code by successively entering

```
get(hf,'Type')
get(ha,'Parent')==hf
P=get(ha,'Position');
set(ha,'Position',[P(1) P(2)+P(4)/4 P(3) P(4)/2]);
set(hf,'Color',[0 0 0]);
set(h,'Color',[1 0 0],'Linewidth',5);
hf1=figure; ha1=axes('Parent',hf1);
set(h,'Parent',ha1);
```

Observe what happens when each of the above lines is executed and comment on the outcome.

Properties of objects are used not only to control their appearance but also their behaviour and functionality.

Exercise 3. Enter the lines:

```
hf=figure;
set(hf,'HandleVisibility','on');
get(hf,'HandleVisibility')
gcf==hf
rootCh=get(0,'Children');
find(rootCh==hf)
set(hf,'HandleVisibility','off');
get(hf,'HandleVisibility')
gcf==hf
rootCh=get(0,'Children');
find(rootCh==hf)
```

[1]These files are provided on a companion CD.

Explain the outcome of the above code. Use MATLAB's help or helpdesk to review the commands `gcf`, `gca` and `gco`. Examine the code of these functions.

Note that if the `HandleVisibility` property of a figure is set to `'off'`, its handle is not a value of the root's `CurrentFigure` property. Similarly, axes with hidden handles do not appear in their parent figure's `CurrentAxes` property, and all figure's children with hidden handles do not appear in its `CurrentObject` property. These "bookkeeping" rules have a direct effect on the results of commands `gcf`, `gca` and `gco`[2]. Objects whose `HandleVisibility` property is set to `'off'` or `'Callback'` are not listed in the `Children` property of their parent. However, an object's `Parent` property is not affected by the visibility of its handle.

Exercise 4. Enter the commands one by one:

```
hf=figure;
hL=plot(rand(10,1),'Tag','MyLine');
```

Click on the resulting line object and verify (by checking the tag and comparing the handles) that this makes the line `hL` the figure's current object:

```
h=get(hf,'CurrentObject')
get(h,'Tag')
disp(h-hL);
```

Now hide the line's handle (`set(h,'HandleVisibility','off')`), click on it again and check what the figure's current object now is (using `gco`).

The following exercise examines the root's `CallbackObject` property.

Exercise 5. Enter:

```
hf=figure('NumberTitle','off');
f=@(h,eD) set(gcbf,'Name',['CallbackObject: ' get(gcbo,'Type')]);
ha=axes('Position',[0.1 0.3 0.8 0.6]);
hL=line([1:10],rand(1,10));
hb=uicontrol('Style','Pushbutton');
hum=uimenu('Label','Test');
set([hf ha hL],'ButtonDownFcn',f);
set([hb hum],'Callback',f);
```

First a few comments on the above code:

- Object creation commands in MATLAB can be invoked with additional arguments specifying the object's Property-Value pairs;
- In this example we use a construct which is described in the helpdesk entry "Anonymous Functions";
- The reason why we used this particular form of input arguments will be discussed later;
- Note that vectors of handles can be used in `set` and `get` commands provided each of the objects listed have the corresponding property and accepts the input value used.

[2]Also: `findobj`, `newplot`, `cla`, `clf` and `close`.

Now click on any object created (figure, axes, line, push button, or the menu labelled “Test”) and observe what happens with the name of the figure.

Exercise 6. Consider Exercise 5 with variable `f` defined as follows:

```
f=@(h,eD) set(gcbf,'Name',['CallbackObject: ' get(h,'Type')]);
```

Some properties of MATLAB’s objects use for their values references to functions or programs to be executed. In Example 5 those were `Callback` and `ButtonDownFcn`. If the data type used as a value for such properties is a *function handle*, then the first argument of the associated function must be treated by its code as a callback object handle and the second argument as an event data. Many of the events in MATLAB do not have their event data implemented yet, so an empty array is passed instead. The following example uses an event type (`KeyPressFcn`) for which the event data is non-empty.

Exercise 7. The function used in this exercise can be found in the folder `Section1` of examples. The code for this function (file `eDtest.m`) is provided below.

```
function eDtest
F.Units='characters';
F.Position=[40 20 60 28];
F.MenuBar='none';
A.Units='normalized';
A.Position=[0 0 1 1];
A.Color=[0 0 0];
T.HorizontalAlignment='center';
T.Color=[0 1 0];
T.Interpreter='none';
S=char('c'*ones(20,20));
hf=figure(F);
ha=axes(A);
hT=text(0.5,0.5,S,T);
set(hf,'UserData',hT,'KeyPressFcn',@eDtestF);

function eDtestF(h,eD)
hT=get(h,'UserData');
if length(eD.Character)>0
    S=char(ones(20,20)*eD.Character);
else
    S='';
end
set(hT,'String',S);
```

- Property/value pairs can be specified using structures;
- Note the use of the figure’s `UserData` property for storing the handle of the text-object;
- In this example we use the field `Character` of the `KeyPressFcn` event data structure.

2 Graphics Object Hierarchy

Graphical Objects in MATLAB are arranged in a structure called the Graphics Object Hierarchy which can be viewed as a *rooted tree* with the nodes representing the objects and the root vertex corresponding to the root object. The form of such a tree depends on the particular set of objects initialised and the precedence relationship between its nodes is reflected in the object's properties `Parent` and `Children`. If for example, the handle `h` of object A is the value of the property `Parent` of object B, then A is the parent of B, and, accordingly, B is a child of A.

Exercise 8. Consider the following code (see file `exerc8.m`).

```
hf=figure('MenuBar','none');
hTb=uitoolbar;
hPT=uipushtool(hTb,'CData',zeros(8,8,3));
hTT=uitoggletool(hTb,'CData',rand(8,8,3),'Separator','on');
hM=uimenu(hf,'Label','Test');
ha=axes('Position',[0.45 0.1 0.5 0.8]);
hL(1)=plot([0:10],rand(11,1));
hL(2)=line([0:10],sin(pi*[0:10]/10),'Color',[0 1 0]);
hL(3)=annotation('line',[0.75 0.75],[0 1],'Color',[1 0 0]);
ha1=get(hL(3),'Parent');
hGO(1)=rectangle('Position',[4,1.5,1,0.2],'FaceColor',[0 0.5 0]);
hGO(2)=line([4 5],[1.5 1.7],'Color',[1 0 0]);
hG=hggroup;
set(hGO,'HitTest','off','Parent',hG);
PanProp.Parent=hf;
PanProp.Position=[0.05 0.1 0.3 0.8];
PanProp.Title='UI panel example';
PanProp.BackgroundColor=get(hf,'Color');
hPan=uipanel(PanProp);
UIPr.Units='normalized';
UIPr.Parent=hPan;
UIPr.Position=[0.1 0.05 0.8 0.07];
UIPr.Style='pushbutton';
hUI(1)=uicontrol(UIPr,'String','Test Button');
UIPr.Style='listbox';
UIPr.Backgroundcolor=[1 1 1];
UIPr.Position=[0.1 0.17 0.8 0.75];
hUI(2)=uicontrol(UIPr,'String',fieldnames(get(hf)));
```

After we have run the script the following objects are created:

- figure `hf`;
  - UI objects: UI toolbar `hTb` and UI push/toggle tools (`hPT` and `hTT`, children of `hTb`), UI menu `hM`, UI panel `hPan`, UI controls `hUI(1)` and `hUI(2)` (styles `pushbutton` and `listbox` respectively);

- Axes `ha` and its children:
  - A core object line `hL(1)`;
  - A plot object lineseries `hL(2)`;
  - A group object `hG` parenting the core objects `hGO(1)` and `hGO(2)` (type rectangle and line respectively);
- Hidden annotation axes `ha1` and its child:
  - annotation type line `hL(3)`.

A general form of the resulting hierarchy is shown in Figure 1.

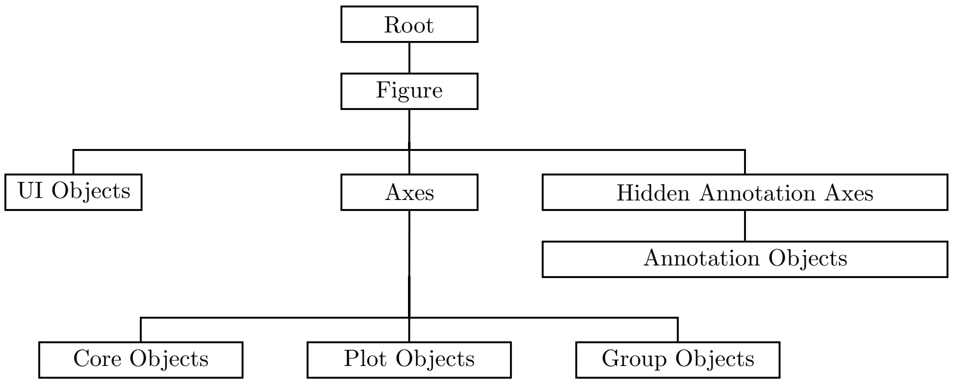


Figure 1: Graphics Object Hierarchy for Exercise 8

A different set of objects can result in a different containment hierarchy.

Exercise 9. The function in this exercise (see file `exerc9.m`) creates a set of objects (see Figure 2) which are organized in a graphics hierarchy somewhat different from the one shown in Figure 1.

```
function exerc9
C=[0.9 0.9 0.8];
hf=figure('MenuBar','none','Color',C,'NumberTitle','off');
hp1=uipanel('Position',[0.1 0.1 0.8 0.8],...
    'Title','UI Panel 1','BackgroundColor',C);
ha=axes('Parent',hp1,'Position',[0.4 0.1 0.5 0.8]);
hT=text(0.5,0.5,'','HorizontalAlignment','center',...
    'FontWeight','bold');
hp2=uipanel('Parent',hp1,'Position',[0.1 0.1 0.2 0.8],...
    'Title','UI Panel 2','BackgroundColor',C);
H=1/13;
for k=1:6
    Prop.Parent=hp2;
```

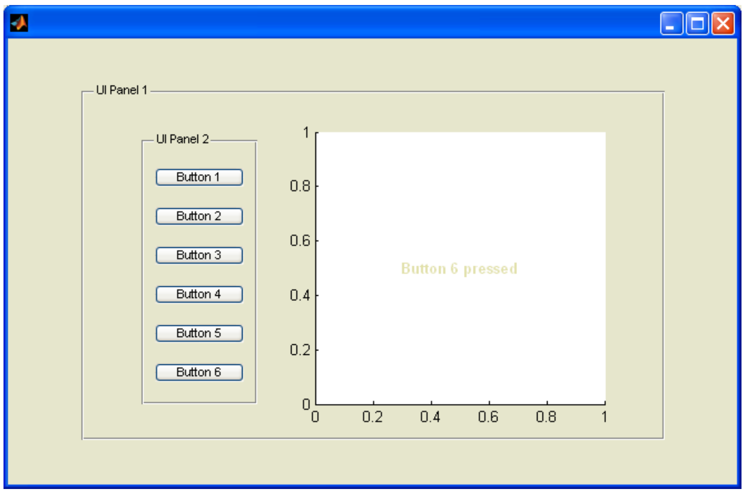


Figure 2: Objects of Exercise 9

```
        Prop.Style='pushbutton';
        Prop.Position=[0.1 H*(2*k-1) 0.8 H];
        Prop.String=['Button ' num2str(7-k)];
        strTxt=[Prop.String ' pressed'];
        Prop.Callback={@bcbF hT strTxt};
        uicontrol('Units','normalized',Prop);
    end

    function bcbF(h,eD,hT,strTxt)
    set(hT,'String',strTxt);
    set(hT,'Color',0.2+0.8*rand(1,3));
```

The function creates the following objects:

- figure `hf` and its children:
  - UI object `hp1` of type "UIpanel" and its children:
    - UI object `hp2` of type "UIpanel" and its children:
      - Six UI controls of style "pushbutton";
  - axes `ha` and its child of type "text".

A graphical representation of the resulting hierarchy is given in Figure 3.

The possibility to have one or another form of hierarchy depends on the admissible values of objects' properties `Parent` and/or `Children`.

One also has to be aware of some specific rules which apply to the use of particular objects. For example, it is recommended that we do not parent any objects to the annotation axes, and we do not change explicitly annotation axes' properties (e.g. search the Help Desk using the entry "Annotation Layer"). Similarly, parenting annotation objects to standard axes may cause errors: enter the code

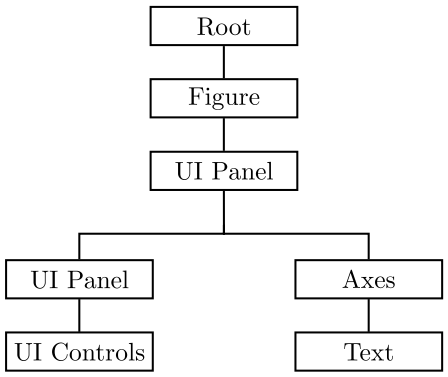


Figure 3: Graphics Object Hierarchy for Exercise 9

```
ha=axes;
h=annotation('line',[0 1],[0 1]);
set(h,'Parent',ha);
```

and see what happens when the figure is resized.

Exercise 10. Show that in MATLAB 7 the annotation axes and standard axes have different set of properties, though the value of their property `Type` is the same string `'axes'`.

This can be done using the following script (file `exerc10.m`):

```
ha=axes;
h=annotation('arrow',[0 1],[0 1]);
haA=get(h,'Parent');
disp(['Type for Standard:   ' get(ha,'Type')]);
disp(['Type for Annotation: ' get(haA,'Type')]);
A=fieldnames(get(ha));
B=fieldnames(get(haA));
[AnB,BnA,inAB]=comp_sc(A,B);
disp('In standard axes, but not in annotation axes:');
disp(A{AnB});
disp('In annotation axes, but not in standard axes:');
disp(B{BnA});
```

Running this code gives the result:

```
Type for Standard:   axes
Type for Annotation: axes
In standard axes, but not in annotation axes:
CurrentPoint
In annotation axes, but not in standard axes:
ObserveFigChildAdded
```

The code of the function `comp_sc` which compares two cells of strings is given below (file `comp_sc.m`).

```
function [AnB,BnA,inAB]=comp_sc(A,B)
[C,inAB.Aind,inAB.Bind]=intersect(A,B);
AnB=setdiff([1:length(A(:))],inAB.Aind);
BnA=setdiff([1:length(B(:))],inAB.Bind);
```

A similar exercise in earlier releases of MATLAB[3], e.g. in Version 6.5.1, would yield a different result. We can also see that the annotation and standard axes have different underlying classes if we continue the script in `exerc10.m` by entering:

```
disp(['Standard: ' class(handle(ha))]);
disp(['Annotation: ' class(handle(haA))]);
```

The following result will be displayed:

```
Standard: axes
Annotation: scribe.scribeaxes
```

The result in MATLAB 6.5.1 would be

```
Standard: axes
Annotation: graph2d.annotationlayer
```

We now consider two more examples of objects which have the same value for their property `Type`.

Exercise 11. The lines which can be created by running the script below (file `exerc11.m`) are different kinds of lines, but the value of the property `Type` is the string `'line'` for each of them.

```
hL1=line([0:1],[0.5 0.5],'Color',[1 0 0]);
hold on;
hL2=plot([0:0.1:1],[0:0.1:1].^2);
%%%%
disp(['Type Value of hL1: ' get(hL1,'Type')]);
disp(['Type Value of hL2: ' get(hL2,'Type')]);
disp(['hL1: ' class(handle(hL1))]);
disp(['hL2: ' class(handle(hL2))]);
%
A=fieldnames(get(hL1));
B=fieldnames(get(hL2));
[AnB,BnA,inAB]=comp_sc(A,B);
A(AnB)
B(BnA)
```

[3]Note that `annotation` is not a standard command in MATLAB 6.5 and the annotation objects in MATLAB 6.5 have to be created via the figure's UI toolbar.

The following answer will be displayed:

```
Type Value of hL1: line
Type Value of hL2: line
hL1: line
hL2: graph2d.lineseries
ans =
   Empty cell array: 0-by-1
ans =
    'DisplayName'
    'XDataMode'
    'XDataSource'
    'YDataSource'
    'ZDataSource'
```

Note that the data source properties of *plot objects* are used by the function `refreshdata`.

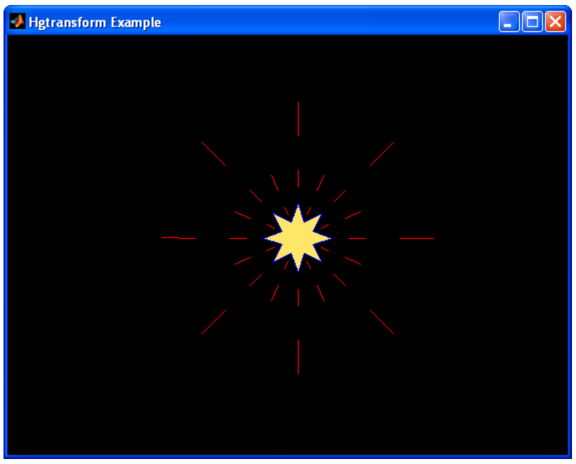


Figure 4: Hgtransform object of Exercise 12

Exercise 12. The code below (file `exerc12.m`) defines a function which creates an `hgtransform` object (see Figure 4) and returns a handle to it. This function exploits the `Matrix` property of hgtransform objects.

```
function hT=exerc12
hf=figure('DoubleBuffer','on','Menubar','none',...
    'NumberTitle','off',...
    'Name','Hgtransform Example');
ha=axes('XLim',[-5,5],'YLim',[-5,5],...
    'DataAspectRatio',[1 1 1],...
    'NextPlot','add');
set([hf, ha],'Color',[0 0 0])
r(1:2:17)=ones(1,9);
r(2:2:16)=0.5*ones(1,8);
```

```
phi=[0:1:16]*(2*pi)/16;
x=r.*cos(phi);
y=r.*sin(phi);
hP=patch(x,y,[1 0.9 0.4]);
hL=line(x,y,'Color',[0 0 1]);
hLP1=line([3*x;4*x]/2,[3*y;4*y]/2,'Color',[1 0 0]);
hLP2=line([3*x;4*x],[3*y;4*y],'Color',[1 0 0]);
hT=hgtransform;
set([hP; hL; hLP1; hLP2],'HitTest','off','Parent',hT);
set(hT,'ButtondownFcn',@spinStar);

function spinStar(h,eD)
hT=gcbo;
for k=1:1:30
    pause(0.1);
    set(hT,'Matrix',makehgtform('zrotate',pi*k/15));
end
```

Now run the script

```
hT1=exerc12;
hT2=annotation('line',[0.1 0.1],[0 1],'Color',[1 0 0]);
disp(['Type Value of hT1: ' get(hT1,'Type')]);
disp(['Type Value of hT2: ' get(hT2,'Type')]);
disp(['hT1: ' class(handle(hT1))]);
disp(['hT2: ' class(handle(hT2))]);
%
A=fieldnames(get(hT1));
B=fieldnames(get(hT2));
[AnB,BnA,inAB]=comp_cells(A,B);
A(AnB)
B(BnA)
```

The following answer will be displayed:

```
Type Value of hT1: hgtransform
Type Value of hT2: hgtransform
hT1: hgtransform
hT2: scribe.line
ans =
    'Selected'
ans =
    'LineWidth'
    'LineStyle'
    'Color'
    'X'
    'Y'
```

The general aim of Exercises 8 to 12 was to provide some simple examples illustrating the use of objects associated with the `Axes` and `Hidden Annotation Axes` (see Figure 1), as well as to explore how they are related. Since the group objects and the annotation objects became standard user-level concepts only in Release 14 we dwelled on them longer. Helpdesk search entires `Core Objects`, `Plot Objects`, `Group Objects` and `Annotation Objects` can be used to access the full specification of all objects in these categories.

UI Objects were also present in some of the considered examples. We will discuss them in more detail in the next section.

3 User Interface Objects

The following objects are grouped under the category of MATLAB User Interface Objects:

- UI controls: check boxes, editable text fields, list boxes, pop-up menus, push buttons, radio buttons, sliders, static text boxes, toggle buttons.
- UI toolbar (`uitoolbar`), which can parent objects of type `uipushtool` and `uitoggletool`;
- UI context menus and UI menus (objects of type `uicontextmenu` and `uimenu`);
- Container objects `uipanel` and `uibuttongroup`.

3.1 UI controls

UI control objects have the same value (string `uicontrol`) for their `Type` property. Their parent can be a figure, a UI panel, or a UI button group. By specifying the `Style` property, we decide whether the resulting object is a check box, a push button, or any other member of the available set of UI controls. The `Style` property can be set to one of the following strings:

- `checkbox`
- `edit`
- `listbox`
- `popupmenu`
- `pushbutton`
- `radiobutton`
- `slider`
- `text`
- `toggle`
- `frame`

Exercise 13. Run the following script and inspect UI controls and their properties in each of the created figures.

```
StyleStr={'checkbox' 'edit' 'listbox' 'popupmenu' ...
    'pushbutton' 'radiobutton' 'slider' 'text' ...
    'toggle' 'frame'};
fcbF=@(h,eD) inspect(findobj(h,'Type','uicontrol'));
```

```
for k=1:length(StyleStr)
    P.Style=StyleStr{k};
    P.Units='pixels';
    switch P.Style
        case {'listbox' 'popupmenu' 'edit'}
            P.String={'Opt. 1' 'Opt. 2' 'Opt. 3' 'Opt. 4'...
                'Opt. 5' 'Opt. 6' 'Opt. 7'};
            P.Position=[50 20 150 60];
            P.Max=2;
            P.HorizontalAlignment='left';
        case 'slider'
            P.String='';
            P.Position=[50 50 150 20];
        otherwise
            P.String='Option 1';
            P.Position=[50 50 150 20];
    end
    hf=figure('MenuBar','none','NumberTitle','off',...
        'Color',[0.6 0.7 0.7],'ButtonDownFcn',fcbF,...
        'Units','pixels','Position',[200 300 250 110]);
    P.Parent=hf;
    P.BackgroundColor=[1 1 1];
    set(hf,'Name',['UI control style: ' P.Style]);
    uicontrol(P);
    clear P;
end
```

The property inspector for each UI control of this example can be invoked by clicking on the corresponding figure window, each of which has its `ButtonDownFcn` set to the anonymous function `fcbF` defined at the beginning of the script. To set the `Callback` property of each UI control to some permissible value, we will use the string `'get(gcbo,''Value'')'`. We can set the callback either via the property inspector by typing `get(gcbo,'Value')` into the `Callback` field for every UI control, or we can do it for all UI controls by entering the commands:

```
h=findobj('Type','uicontrol');
set(h,'Callback','get(gcbo,''Value'')');
```

Once the callback has been set perform some manipulations with the associated UI control. By observing the result displayed in the command window try to find out

- what actions trigger the execution of the callback;
- how the displayed value is connected with the state of the UI control and with its `Min` and `Max` properties.

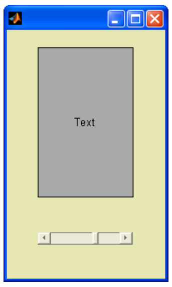
Note that the callbacks specified as strings are executed in the base workspace. How can we use callbacks and events associated with them? Take for example a slider. If the callback function has access to the handle of the slider, we can obtain its `Value` property, and then use the result to reset some other parameter. How can the handle of the control be passed to the callback? If the callback is specified as a string we use the function `gcbo`, which returns the object that was the source of the event. If we specify the callback as a function handle, we just have to keep in mind that in this case the first argument of the callback function is the handle to the source object of the event and the second argument is the event data, and their values are passed to the callback function automatically (also, see our comments on Exercise 6).

Exercise 14. Examine how the callback property of the slider in the code below (file `testSlider.m`) is defined. Here the parameter changed by the associated callback function is the colour of axes.

```
function testSlider
FP.Units='pixels';
FP.Position=[200 200 200 300];
FP.Color=[0.9 0.9 0.7];
FP.Numbertitle='off';
FP.Menubar='none';
hf=figure(FP);
AP.Units='pixels';
AP.Position=[40 100 120 180];
AP.Color=[0 0 0]; AP.Box='on';
AP.XTick=[];      AP.YTick=[];
AP.XLim=[0 1];    AP.YLim=[0 1];
ha=axes(AP);
UiP.Style='slider';
UiP.Units='pixels';
UiP.Position=[40 42 120 15];
UiP.Callback={@sliderCBF ha};
UiP.Max=100;
hS=uicontrol(UiP);
text(0.5,0.5,'Text','HorizontalAlignment','center');
set([hf ha hS],'Units','normalized');
set(hf,'HandleVisibility','off');

function sliderCBF(h,eD,ha)
V=get(h,'Value');
MaxSV=get(h,'Max');
set(ha,'Color',[1 1 1]*V/100);
```

We will use the simple functionality of this example more than once. In Section 4.3 it will illustrate how the GUI interacts with the application m-file and the callback subfunctions. We will also use it to show how Java and ActiveX components can be included in our MATLAB GUI applications.



3.2 Menus, push/toggle tools, and toolbars

The default value of figure's property `MenuBar` is `'figure'`. If we create an empty figure (e.g. by entering `hf=figure;`) with default settings we can see that there are two rows of user interface objects in the upper part of the figure. These objects are of type `uimenu` (located in the upper row), and of type `uipushtool` and `uitoggletool` (in lower row). These `uipushtool` and `uitoggletool` objects are contained within and parented by a bar-like object whose type is `uitoolbar`.

The following exercise illustrates the usage of uipushtools and uitoggletools.

Exercise 15. Create a standard empty figure and suggest how we could access and extract the `CData` information of its uipushtools and uitoggletools.

One possible solution is given by the the following code (file `exerc15.m`).

```
hf=figure;
hT=findall(hf,'Type','uitoolbar');
h=findall(hT,'Parent',hT);
hT1=uitoolbar(hf);
hPT=uipushtool(hT1);
f=@(hObj,eD) set(hPT,'CData',get(hObj,'CData'));
set(h,'ClickedCallback',f,'Enable','on');
```

A few comments on this code.

- Here we used the command `findall` to obtain the handles of the `uitoolbar` and UI-push/toggle-tools. Alternatively, we could have used

  ```
  hT=findobj(hf,'Type','uitoolbar');
  h=get(hT,'Children');
  ```

  after setting the root's property `ShowHiddenHandles` to `'on'`.

- Note that the callbacks for the `uipushtool` and `uitoggletool` are specified in the field `ClickedCallback`. As we did in a number of previous examples, an anonymous function was used in order to make the code more compact.

- The last line in the script resets the `ClickedCallback` property of the UI-push/toggle-tools in the uitoolbar `hT`, so that pushing any of them results in appropriately resetting the `CData` property of the `uipushtool` parented by the uitoolbar `hT1`.

3.2.1 UI menus and context-menus

UI menus can be parented by figure objects, other UI menus and the objects of type `contextmenu`. The following example illustrates these options.

Exercise 16. Examine the following script (file `exerc16.m`) and run it.

```
hf=figure('MenuBar','none','NumberTitle','off','Name','UI-menus');
```

```
N=3; NN=3; NNN=3;
for k=1:N
    h(k)=uimenu(hf,'Label',['menu-' num2str(k)]);
    for m=1:NN
        L=['menu-' num2str(k) '.' num2str(m)];
        hh(k,m)=uimenu(h(k),'Label',L);
        if m>1
            set(hh(k,m),'Separator','on');
        end
        for q=1:NNN
            L=['menu-' num2str(k) '.' num2str(m) '.' num2str(q)];
            hhh(k,m,q)=uimenu(hh(k,m),'Label',L);
            if q>1
                set(hhh(k,m,q),'Separator','on');
            end
        end
    end
end
set(hhh(:),'Callback','set(gcbf,''Color'',rand(1,3))');
%%
t=[1:100]*4*pi/100;
hL=plot(t,sin(t),'Color',[0 0 0],'Linewidth',4);
f=@(h,eD,hP,C) set(hP,'Color',C);
hCm=uicontextmenu;
hR=uimenu(hCm,'Label','Red','Callback',{f hL [1 0 0]});
hG=uimenu(hCm,'Label','Green','Callback',{f hL [0 1 0]});
hB=uimenu(hCm,'Label','Blue','Callback',{f hL [0 0 1]});
set([hG hB],'Separator','on');
set(hL,'UIcontextmenu',hCm);
```

Comments on this script:

- UI-menus `h(k)` labelled as `menu-k` are children of the figure `hf`, ui-menus `hh(k,m)` labelled as `menu-k.m` are childern of the ui-menu `h(k)`, and ui-menus `hhh(k,m,q)` labelled as `menu-k.m.q` are parented by the ui-menu `hh(k,m)`.

- The callback function of ui-menus `hhh(k,m,q)` resets the color of the figure `hf` to some randomly chosen color.

- UI-menus `hR`, `hG` and `hB`, labelled as `Red`, `Green` and `Blue` are parented by the ui-contextmenu `hCm`, these ui-menus can be seen if we right-click on the plot line `hL`. The callback functions of these ui-menus reset the color of the line `hL` to red, green and blue respectively.

- The ui-contextmenu `hCm` is associated with the object `hL` by setting the `UIContextMenu` property of `hL` to `hCm`.

3.3 Container objects: uipanel and uibuttongroup

Both `uipanel` and `uibuttongroup` have the same value of their type property, the string `'uipanel'`. They also have the same set of properties which control their appearance. We can use the function `comp_cells` defined in Section 2 to find that `uibuttongroup` objects have additional properties `SelectionChangeFcn` and `SelectedObject`.

Exercise 17. Run the function `exerc17` (file `exerc17.m`) whose code is shown below.

```
function exerc17
P=[20 20 100 15]; P1=[0 20 0 0];
hf=figure;
h1(1)=uicontrol('Style','radiobutton','Position',P);
h1(2)=uicontrol('Style','togglebutton','Position',P+P1);
h1(3)=uicontrol('Style','radiobutton','Position',P+2*P1);
hBG=uibuttongroup('Position',[0.5 0.1 0.4 0.5]);
h2(1)=uicontrol('Parent',hBG,'Style','radiobutton','Position',P);
h2(2)=uicontrol('Parent',hBG,'Style','togglebutton','Position',P+P1);
h2(3)=uicontrol('Parent',hBG,'Style','radiobutton','Position',P+2*P1);
set(h2(1),'String','Select-1');
set(h2(2),'String','Select-2','BackgroundColor',[0.4 0.4 0.4]);
set(h2(3),'String','Select-3');
set(hBG,'SelectionChangeFcn',@bgCbck);

function bgCbck(h,eD)
hUi=get(h,'SelectedObject');
disp(['Selected control: ' num2str(hUi) ', ' get(hUi,'String')]);
disp('Event-data structure');
disp(eD);
```

- We have two groups of ui-controls (radio- and toggle-buttons) here. Note that the group parented by the figure does not possess the exclusive behaviour, i.e. all combinations of ui-controls in this group can be selected simultaneously, whereas only a single selection is possible for the group parented by the `uibuttongroup`.
- The callback function for the `SelectionChangeFcn` property was designed to show the handle and the `string` property of the selected ui-control and to display the event data structure passed as the second argument. The field names of the event data structure are `EventName` (the value associated with this field is the string `'SelectionChanged'`), `OldValue` and `NewValue`.

Exercise 18. This exercise uses `uipanel` objects (see Figure 5). The code contained in the files `exerc18.m` and `mPan.m` is shown below.

File `exerc18.m` code:

```
hf=figure('HandleVisibility','off',...
    'NumberTitle','off','Name','Stacked plots',...
    'MenuBar','none');
F={@sin @tan @exp @log};
```

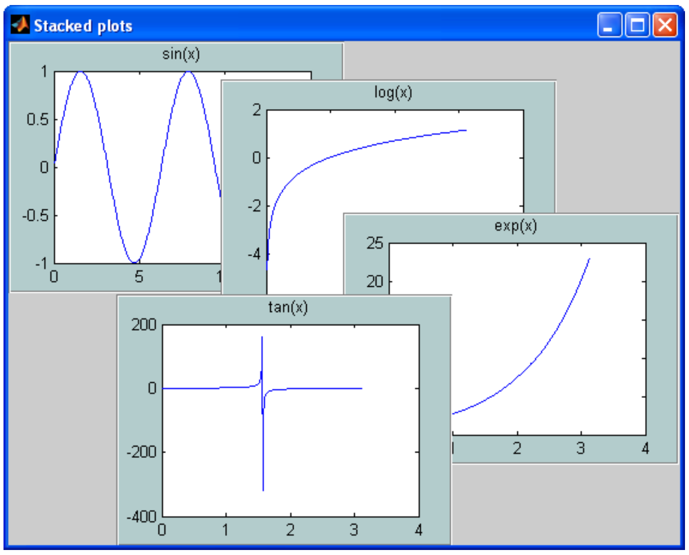


Figure 5: Exercise 18: uipanel objects as parents of axes

```
x=[0.003:0.003:1]*pi;
X={4*x,x,x,x};
strF={'sin(x)' 'tan(x)' 'exp(x)' 'log(x)'};
for k=1:length(F)
    hP=mPan(hf);
    axP.Parent=hP;
    ha=axes(axP);
    plot(X{k},F{k}(X{k}),'Parent',ha);
    set(get(ha,'Title'),'String',strF{k});
end
```

File `mPan.m` code:

```
function varargout=mPan(hf)
PanPr.BackgroundColor=[0.7 0.8 0.8];
PanPr.Position=[0 0.5 0.5 0.5];
PanPr.ButtonDownFcn=@panBDF;
PanPr.Parent=hf;
h=uipanel(PanPr);
set([hf h],'Units','pixels');
if nargout==1
    varargout{1}=h;
end

function panBDF(h,eD)
hf=ancestor(h,'figure');
hfCh=get(hf,'Children');
vI=find(~(hfCh==h));
set(hf,'Children',[h;hfCh(vI)]);
```

```
refresh(hf);
xy0=get(hf,'CurrentPoint');
P0=get(h,'Position');
set(hf,'WindowButtonMotionFcn',{@panWBMF h xy0 P0});
set(hf,'WindowButtonUpFcn',@panWBUF);

function panWBMF(hf,eD,h,xy0,P0)
xy=get(hf,'CurrentPoint');
dxy=xy-xy0;
P=get(h,'Position');
Pf=get(hf,'Position');
set(h,'Position',[min(max(P0(1)+dxy(1),0),Pf(3)-P0(3)),...
    min(max(P0(2)+dxy(2),0),Pf(4)-P0(4)), P0(3), P0(4)]);

function panWBUF(hf,eD)
set(hf,'WindowButtonMotionFcn','');
set(hf,'WindowButtonUpFcn','');
```

- Here we create movable `uipanel` objects, and then use them to parent axes with various plots.
- Each `uipanel` with the required properties is created by the function `mPan`.
- Auxiliary functions `panBDF`, `panWBMF` and `panWBUF` are used as callback functions for:
  - the `ButtonDownFcn` property of the `uipanel` object,
  - the `WindowButtonMotionFcn` of the figure, once the button has been pressed over the panel and before it has been released,
  - the `WindowButtonUpFcn` of the figure, once the button has been pressed over the panel and before it has been released.

4 Programming MATLAB GUI

In this section we will discuss some concepts and techniques needed to program GUIs. Typically we would like a GUI to respond in an appropriate manner to events resulting from the interaction of the user with one or another component of the GUI. In a common scenario, after some specific graphical interaction with the UI component, a corresponding event is triggered and a callback function is executed. The reference to the callback function is passed using the relevant component's property (such as `Callback`, `ClickedCallback`, `ButtonDownFcn`, etc.).

In many cases the actions performed by the callback depend on the state of the GUI, so a standard mechanism is required which would allow each callback to routinely access properties of objects. Data exchange must also be properly organised.

A GUI writer must be aware of certain MATLAB language conventions. For example, if the callback property is specified as a string, then the callback expression is evaluated in the base workspace, but if it is specified as a cell-array, then the variables additional to the callback object handle and the event data will be interpreted in the scope of the segment of the code where the callback property was originally set.

4.1 Some useful constructs and concepts

We now review some useful constructs and functions.

4.1.1 Function `feval`

If a function `F` (with the code in the file `F.m`) accepts input arguments `x1`, `x2`, . . . , `xN` and returns output values for `v1`, `v2` and `vM`, we call it using `[v1,v2,...,vM]=F(x1,x2,...,xN)`. We can also do it using either

```
[v1,v2,...,vM]=feval(@F,x1,x2,...,xN)
```

or

```
[v1,v2,...,vM]=feval('F',x1,x2,...,xN)
```

As an example compare the result of each of the following three assignments

`y=sin(pi)`, `y=feval(@sin,pi)` and `y=feval('sin',pi)`.

4.1.2 Constructs `varargin` and `varargout`

We use the `varargin` and `varargout` constructs if we wish to make a function accept a variable number of input and output arguments and behave according to the number of the input or output arguments used. When used, `varargin` (`varargout`) must appear as the last input (output) argument in the function specification where both `varargin` and `varargout` are treated as cell-arrays.

Often `varargin` and `varargout` are used together with `nargin` and `nargout`.

Exercise 19. Examine the following code:

```
function varargout=varnargtest(C,varargin)
x=[0:500]*4*pi/500;
hf=figure('NumberTitle','off','Name','Test varargin/varargout');
disp(['Number of input args.:  ' num2str(nargin)]);
disp(['Number of output args.: ' num2str(nargout)]);
switch nargin
    case 1
        h=plot(x,rand(1,length(x)),'Color',C);
    otherwise
        f=varargin{1};
        h=plot(x,feval(f,x),'Color',C);
end
if nargout>0
```

```
        varargout{1}=hf;
        if nargout>1
            varargout{2}=h;
        end
    end
```

The following lines can be used to test this function:

```
    varnargtest([0 0 1]);
    varnargtest([0 0 1],@sin);
    [hf,h]=varnargtest([0 0 1],@sin,1,1,1,1,1,1);
```

4.1.3 Replacing Lists of Variables with Cell Arrays

Cell arrays used in function input or output lists (with the colon to index multiple cells) are interpreted by MATLAB as comma-separated lists of variables, each cell being treated as a separate variable.

Exercise 20. The code below can be found in the file `exerc20.m`.

```
    % Colon-indexed cell arrays as input arguments
    x=4*[0:0.01:1]*pi;
    v={x sin(x) 'Color' [0 1 0]};
    plot(v{:});
    plot(v{1:2});
    % Colon-indexed cell arrays as output arguments
    g=rand(10,1);
    [M{1:2}]=min(g);
    disp(M{1}-g(M{2}));
```

Exercise 21. There are cases when we would like to be able to access and use subfunctions of a main function by calling it with appropriate arguments. Use of colon-indexed cell arrays in conjunction with `feval` can facilitate this. The following code is in the file `Section4/tstfcn.m`.

```
    function varargout=tstfcn(varargin)
    if nargin && ischar(varargin{1})
        if nargout
            [varargout{1:nargout}]=feval(varargin{:});
        else
            feval(varargin{:});
        end
    else
        S='Just the main function called';
        if nargout
            varargout{1}=S;
        end
        disp(S);
```

```
end

function S=f1(varargin)
S='Subfunction f1 called';
disp(S); disp(['Number of input arguments: ' num2str(nargin)]);

function S=f2(varargin)
S='Subfunction f2 called';
disp(S); disp(['Number of input arguments: ' num2str(nargin)]);
```

- Test `tstfcn.m` by successively entering the lines:

  ```
  S=tstfcn
  S=tstfcn('f1')
  S=tstfcn('f2',1,1,1)
  ```

  Examine and explain the output.

- The mechanism used in Exercise 21 allows one to define as a subfunction of `tstfcn` any function `f` with input arguments `v1`, `v2`, ..., `vN` and output arguments `w1`, `w2`, ..., `wM`, and call it using

  ```
  [w1,w2,...,wM]=tstfcn('f',v1,v2,...,vN)
  ```

- Try this syntax to call any standard function or a function on your MATLAB path, e.g. `y=tstfcn('sin',pi)`.

4.2 Data storage and exchange using application data

A typical GUI may consist of several graphical elements. When their associated callback functions process triggered events they often have to use common data sets. One option here is to provide access to such data using global variables. However, use of global variables in callbacks can create situations when functions of one GUI interfere with the workspace or functions of another GUI.

Objects' `UserData` property can hold data of a general nature and allows the GUI writer to avoid defining global variables. Prior to MATLAB 6, GUI figure's property `UserData` was commonly used for data exchange and also storing the handles of objects initialised by the GUI.

Application data was introduced in MATLAB 6. Since `ApplicationData` is not an ordinary property, it is handled by separate functions `setappdata` and `getappdata`. Use of application data for storing and retrieving data associated with the GUI is now a recommended method. The following example shows a reason why the use of application data is preferential to the use of `UserData`.

Exercise 22. Create axes `ha` and store a string `'test'` in its `UserData`:

```
ha=axes; set(ha,'UserData','test');
```

Check that `UserData` of axes `ha` now holds this string:

```
        get(ha,'UserData')
    ans =
          test
```

Now enter

```
    plot(rand(10,1),'Parent',ha);
```

and check `UserData` of axes `ha` again:

```
       get(ha,'UserData')
    ans =
          []
```

What happens here is that, when the `NextPlot` property of axes is set to the string `'replace'`, the command `plot` resets all axis properties, except `Position`, to their default values[4].

Now let us verify that data stored in the application data is not being changed or lost:

```
       ha=axes; setappdata(ha,'MyField','test');
       getappdata(ha,'MyField')
    ans =
          test
       plot(rand(10,1),'Parent',ha);
       getappdata(ha,'MyField')
    ans =
          test
```

4.2.1 Functions `guidata` and `guihandles`.

The function `guidata` is used to store and retrieve application data. It uses the field `UsedByGUIData_m` of the application data structure of the GUI's figure. When `guidata` is called as `guidata(h,D)`, where `h` is a handle of any of the figure's descendants, it stores data `D` in the figure's application data field `UsedByGUIData_m`. If we wish to retrieve data `D` we use the syntax `D=guidata(h)`. The function `guihandles` is often used in conjunction with `guidata`. The command to call `guihandles` is `handles=guihandles(h)`. If `hf` is the figure parenting `h`, then the output of `guihandles(h)` is a structure containing the handles of all descendants of `hf` with non-empty valid tags, the tags being used as the name of fields where the corresponding handles are stored[5].

Exercise 23. The following code is a simple test of the functions `guidata` and `guihandles`. First enter

```
    hf=figure('MenuBar','none');
    ha=axes('Tag','axes1');
    x=[1:500]/500;
```

[4]A few other commands e.g. `surf`, `surfc`, `surfl`, behave similarly.

[5]If a UI-control with a non-empty tag is used as a proxy for an ActiveX control, then the handle of the UI-control is replaced by the handle `ho` of the com-object.

```
h1=line(x,sin(6*pi*x)./x,'Tag','L1');
h2=line(x,cos(6*pi*x)./(x+1),'Color',[1 0 0]);
handles=guihandles(hf);
```

Now verify that `handles` is a structure with the fields `axes1` and `L1` and confirm that `handles.axes1` and `handles.L1` are handles of the axes and the first of the two lines by modifying their properties, e.g.

```
set(handles.L1,'Linewidth',11);
set(handles.axes1,'Visible','off');
```

Use `disp(getappdata(hf))` to check that at this stage the application data of `hf` is a 1×1 structure array with no fields. Continue by entering

```
guidata(hf,handles);
D=getappdata(hf);
H=guidata(hf);
```

Show that `D.UsedByGUIData_m` and `H` hold identical data, a structure with fields `axes1` and `L1` containing the handles `ha` and `h1` of the axes and the first line.

4.3 Callbacks in GUIDE-generated application m-files

The MATLAB GUI Development Environment (GUIDE) allows the user to build a GUI with its Layout Editor. GUIDE also automatically generates a template for the GUI's application m-file which controls the behaviour and functionality of the elements of the GUI. MATLAB's helpdesk provides detailed information on how to use GUIDE, e.g. run `doc('guide')` to access the relevant help pages and links[6]. Here we will consider a few programming aspects of working with the application m-file generated by GUIDE.

Exercise 24. Open GUIDE, choose `Create New GUI`. Now click on the button with the icon showing axes and the tool-tip string `Axes`, then allocate an area for the axes. Similarly, click on the button for the slider and select an area for the slider. Insert a push button in the same way. Save the result to some file, e.g. use `tst` for the name, then view the resulting m-file `tst.m`.

- In the file `tst.m`, find the line

  ```
  function slider1_Callback(hObject, eventdata, handles)
  ```

  At this stage the body of the subfunction `slider1_Callback` is empty, so it does not do anything. To make this subfunction respond in some way to the changes of the state of the slider add the following lines:

  ```
  ha=handles.axes1;
  v=get(handles.slider1,'Value');
  set(ha,'Color',[v v v]);
  ```

[6]Also check the printable manual [1].

Insert the command `close(handles.figure1)` into the definition of the subfunction `pushbutton1_Callback`.

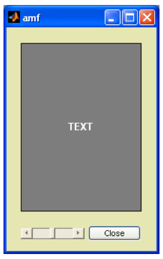
- In GUIDE, double click on the slider area to invoke the Property Inspector. Use it to set the `Value` property of the slider to 1.0. Similarly, set the `String` property of the push button to `Close`. The Property Inspector for each object can also be invoked using the Object Browser. Save changes, run `tst.m` and check that both the slider and the button in the resulting GUI now respond to our actions.
- The GUI built using GUIDE requires the file `tst.fig`. GUIDE allows the user to export the code for the application into an m-file which carries out the GUI figure initialisation programmatically.
- Examine how the function `gui_mainfcn.m` handles the initialisation of the GUI and its callbacks. The callbacks in the following example are organised in a simpler, but similar way. Here the file `amf.m` plays the role of the application m-file and the function `gui_mf` imitates `gui_mainfcn`.

Exercise 25. The following code can be found in the file `Section4\amf.m`

```
function varargout = amf(varargin)
gui_State.gui_Callback=[];
gui_State.gui_LayoutFcn=@amf_fig;
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end
if nargout
    [varargout{1:nargout}] = gui_mf(gui_State, varargin{:});
else
    gui_mf(gui_State, varargin{:});
end

function slider1_Callback(hObject, eventdata, handles)
ha=handles.axes1;
v=get(handles.slider1,'Value');
set(ha,'Color',[v v v]);

function pushbutton1_Callback(hObject, eventdata, handles)
close(handles.figure1);
```



The function `amf_fig` creates the figure with axes, a slider and a pushbutton with the following tags: `figure1`, `axes1`, `slider1` and `pushbutton1`. The callbacks for the slider and the push button are specified as the strings:

```
amf('slider1_Callback',gcbo,[],guidata(gcbo))
amf('pushbutton1_Callback',gcbo,[],guidata(gcbo))
```

File `amf_fig.m` can be found in the same folder `Section4`.

The code for the function `gui_mf` is as follows:

```
function varargout = gui_mf(gui_State, varargin)
numargin = length(varargin);
if numargin == 0
    gui_Create = 1;
elseif ischar(varargin{1}) && numargin>1 ...
        && isequal(ishandle(varargin{2}), 1)
    gui_Create = 0;
end
if gui_Create == 0
    varargin{1} = gui_State.gui_Callback;
    if nargout
        [varargout{1:nargout}] = feval(varargin{:});
    else
        feval(varargin{:});
    end
else
    if nargout
        varargout{1} = feval(gui_State.gui_LayoutFcn);
    else
        feval(gui_State.gui_LayoutFcn);
    end
end
```

Examine the code for both functions and describe step by step how the GUI initialisation and callback execution is carried out.

4.4 More on callback organisation

Callbacks can be organised in a number of ways. Recall that in MATLAB 7 a callback for an object from the graphics object hierarchy can be specified as:

- a string — when the relevant event occurs, this string will be evaluated in the base workspace;
- a function handle, e.g. `f_h` — in this case MATLAB will invoke the callback `f_h` with the two input arguments: `h`, the handle of the callback object, and `eD`, the event data structure;
- a cell array, e.g. `{f_h, v1, . . . , vN}`, where `f_h` is a function handle[7] and `v1`, . . . , `vN` are variables defined prior to the callback specification — when the event takes place, MATLAB will invoke the function with the following input arguments: `h` (callback object's handle), `eD` (event data structure), and the variables `v1`, . . . , `vN` holding the values they had at the moment of the callback specification.

[7]This can also be the name of a function residing on the MATLAB search path.

Callbacks can be created as separate m-functions residing on the MATLAB search path. For callbacks with large blocks of code this approach would be preferential. However, in the cases when there are many small callbacks it is more convenient to keep their definitions in one file.

In Exercise 26 we consider three coding examples which have the same functionality but use different principles of organisation and dispatch of callbacks. Each of those examples is confined to a single file.

Exercise 26. In the function definition (file `pd_1.m`) below a switchyard technique is used to make it perform both initialisation and callback execution tasks.

```
function pd_1(action)
if nargin==0
    action='initialize';
end
switch(action)
    case 'initialize'
        hf=figure; ha=axes('Parent',hf);
        set(ha,'Tag','SpotA','XLim',[0 1],'YLim',[0 1]);
        set(hf,'DoubleBuffer','on','Tag','SpotF');
        L.Marker='o'; L.MarkerSize=10;
        L.MarkerFaceColor=[0 0 1];
        L.MarkerEdgeColor=[0 0 1];
        L.Tag='Spot';
        L.ButtonDownFcn='pd_1 start';
        handles.Spot=line(0.5,0.5,L);
        handles.SpotF=hf;
        handles.SpotA=ha;
        set(handles.SpotF,'HandleVisibility','off',...
            'UserData',handles);
    case 'start',
        handles=get(gcbf,'UserData');
        set(handles.SpotF,'WindowButtonMotionFcn','pd_1 move');
        set(handles.SpotF,'WindowButtonUpFcn','pd_1 stop');
    case 'move'
        handles=get(gcbf,'UserData');
        xL=get(handles.SpotA,'XLim');
        yL=get(handles.SpotA,'YLim');
        currPt=get(handles.SpotA,'CurrentPoint');
        set(handles.Spot,'XData',...
            min(max(currPt(1,1),xL(1)),xL(2)));
        set(handles.Spot,'YData',...
            min(max(currPt(1,2),yL(1)),yL(2)));
    case 'stop'
        handles=get(gcbf,'UserData');
        set(handles.SpotF,'WindowButtonMotionFcn','');
        set(handles.SpotF,'WindowButtonUpFcn','');
end
```

- This code creates a figure, axes and a movable circle. When the user clicks on the circle, it will follow the motion of the cursor within the limits of the axes until the mouse button is released.
- Consider for example the `ButtonDownFcn` property of the circle. It is specified as `'pd_1 start'`. Running the command `pd_1 start` is equivalent to running `pd_1('start')`[8]. Note that the `WindowButtonMotionFcn` and `WindowButtonUpFcn` are set dynamically and review the comments on Exercise 18 where a similar technique was implemented in a different manner.
- In this example handles are stored in figure's `UserData`.

The technique used in the following example (file `pd_2.m`) was probably introduced by MATLAB 6 GUIDE. Callbacks in this function are being sorted through by means of `feval`[9].

```
function varargout=pd_2(varargin)
if nargin<=1
    hf=figure('DoubleBuffer','on','Tag','SpotF');
    ha=axes('Tag','SpotA','XLim',[0 1],'YLim',[0 1]);
    L.Marker='o';
    L.ButtonDownFcn='pd_2(''StartFcn'',guidata(gcbo))';
    L.MarkerSize=10;
    L.MarkerFaceColor=[0 0 1];
    L.MarkerEdgeColor=[0 0 1];
    L.Tag='Spot';
    L.Parent=ha;
    hL=line(0.5,0.5,L);
    handles=guihandles(hf);
    guidata(hf,handles);
    set(handles.SpotF,'HandleVisibility','off');
    if nargout==1
        varargout{1}=hf;
    end
elseif ischar(varargin{1})
    feval(varargin{:});
end

function StartFcn(handles)
set(handles.SpotF,'WindowButtonMotionFcn',...
    'pd_2(''MoveFcn'',guidata(gcbo))');
set(handles.SpotF,'WindowButtonUpFcn',...
    'pd_2(''StopFcn'',guidata(gcbo))');

function MoveFcn(handles)
```

[8]To verify this, run `docsearch('passing string arguments to functions')`.

[9]Review Exercise 21, which considers how `feval` can be used to access a subfunction of a function.

```
xL=get(handles.SpotA,'XLim');
yL=get(handles.SpotA,'YLim');
currPt=get(handles.SpotA,'CurrentPoint');
set(handles.Spot,'XData',...
    min(max(currPt(1,1),xL(1)),xL(2)));
set(handles.Spot,'YData',...
    min(max(currPt(1,2),yL(1)),yL(2)));

function StopFcn(handles)
set(handles.SpotF,'WindowButtonMotionFcn','');
set(handles.SpotF,'WindowButtonUpFcn','');
```

- Note the use of the combination

  `handles=guihandles(hf); guidata(hf,handles)`

  each time we need to save (or update) the handles structure in the figure's application data, and the use of the command `guidata(gcbo)` when we wish to retrieve the saved handles.

- In GUIDE generated application m-files callback subfunctions have the format:

  `callback_func(hObject,eD,handles,...)`

  where `hObject` is the handle of the callback object, `eD` is an event data structure and `handles` is the handles structure. A different format can be used as long as it complies with the form of the specified callback string.

In the two previous examples the callbacks were specified as strings. In the following example (file `pd_3.m`) the callbacks are set using function handles.

```
function varargout=pd_3
hf=figure('DoubleBuffer','on','Tag','SpotF');
ha=axes('Tag','SpotA','XLim',[0 1],'YLim',[0 1]);
L.Marker='o';
L.MarkerSize=10;
L.MarkerFaceColor=[0 0 1];
L.MarkerEdgeColor=[0 0 1];
L.Tag='Spot';
L.Parent=ha;
hL=line(0.5,0.5,L);
set(hL,'ButtonDownFcn',@StartFcn);
handles=guihandles(hf);
guidata(hf,handles);
set(handles.SpotF,'HandleVisibility','off');
if nargout==1
    varargout{1}=hf;
```

```
end

function StartFcn(hObject,eD)
handles=guihandles(hObject);
set(handles.SpotF,...
    'WindowButtonMotionFcn',{@MoveFcn guidata(gcbo)});
set(handles.SpotF,'WindowButtonUpFcn',@StopFcn);

function MoveFcn(hObject,eD,handles)
xL=get(handles.SpotA,'XLim');
yL=get(handles.SpotA,'YLim');
currPt=get(handles.SpotA,'CurrentPoint');
set(handles.Spot,'XData',...
    min(max(currPt(1,1),xL(1)),xL(2)));
set(handles.Spot,'YData',...
    min(max(currPt(1,2),yL(1)),yL(2)));

function StopFcn(hObject,eD)
handles=guidata(hObject);
set(handles.SpotF,'WindowButtonMotionFcn','');
set(handles.SpotF,'WindowButtonUpFcn','');
```

- Since in `pd_1.m` and `pd_2.m` the callbacks were specified as strings, the main function name had to be mentioned explicitly. In `pd_3.m` the main function name does not appear in the expressions defining the callbacks.
- Invoke `pd_1` or `pd_2`, then change your current working directory to a directory which is not on your MATLAB path. Observe that the callbacks can no longer be executed, explain why this happens. Now try to do the same with `pd_3`. Explain why the functionality of `pd_3` is not affected by the change of the working directory.

Let us use the technique exploited in Exercise 26 to design something that could be more practical. In Exercise 27 we define a graphical object which can be associated with a plot and which the user can interact with in order to extract information about the plot.

Exercise 27. This is an example of a purpose-built control which is made of descendants of axes (two lines combined into an hggroup). To test the example (file `plotslider.m` in the folder `Section4`) first create a plot, e.g.

```
x=6*pi*[0:500]/500; hP=plot(x,sin(x));
```

Now initialise the control by entering `h=plotslider(hP)`. The code for the function `plotslider` is as follows:

```
function varargout=plotslider(hP)
hA=get(hP,'Parent');
hF=ancestor(hP,'figure');
hG=hggroup('Parent',hA);
```

```
    hUICM=uicontextmenu('Parent',hF);
    set(hG,'uicontextmenu',hUICM);
    hUIM=uimenu(hUICM);
    D.hUICM=hUICM;
    D.hUIM=hUIM;
    D.X=get(hP,'XData');
    D.Y=get(hP,'YData');
    D.hP=hP;
    D.hA=hA;
    D.hF=hF;
    yL=get(hA,'YLim'); xL=get(hA,'XLim');
    MProp.HitTest='off';
    MProp.Color=[0.7 0.7 0.7];
    MProp.Linestyle='none';
    MProp.Marker='o';
    MProp.MarkerFaceColor=[0.7 0.7 0.7];
    MProp.Parent=hG;
    MProp.XData=0.5*(D.X(1)+D.X(end));
    MProp.YData=interp1(D.X,D.Y,MProp.XData);
    Xstr=num2str(MProp.XData);
    Ystr=num2str(MProp.YData);
    set(hUIM,'Label',['X=' Xstr ';' 'Y=' Ystr]);
    MProp.DeleteFcn=@delFcn;
    hM=line(MProp);
    D.hM=hM;
    LProp.HitTest='off';
    LProp.Parent=hG;
    LProp.Color=[0.7 0.7 0.7];
    LProp.XData=MProp.XData*[1 1];
    LProp.YData=yL;
    LProp.DeleteFcn=@delFcn;
    hL=line(LProp);
    D.hL=hL;
    setappdata(hG,'LMData',D);
    set(hG,'ButtonDownFcn',@bdwnFcn);
    set(hP,'DeleteFcn',@pDelFcn);
    hGall=getappdata(hP,'usedByPslider');
    setappdata(hP,'usedByPslider',[hGall hG]);
    if nargout
        varargout{1}=hG;
    end
```

This function creates an `hggroup` object `h` associated with a plot `hP` as shown in Figure 6. Data used by the object `h` are packed into structure `D` and stored in the application data of `h` (field `LMData` is created for this purpose). Data structure `D` contains such information as the handles of the parent axes and figure, as well as the handle of the plot `hP`. In addition to responding graphically to the mouse events, the control will also have a simple functionality which will consist in resetting a label of an associated `uimenu`. The handle of

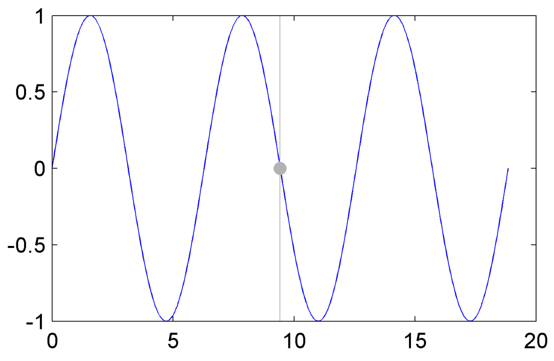


Figure 6: A control associated with the plot

this `uimenu` and the handle of its parent `uicontextmenu` are also stored in the structure `D`. Fields `X` and `Y` of `D` are used to hold `XData` and `YData` of the plot `hP`.

We would like all our controls associated with the plot `hP` to be deleted when this plot is deleted. Since our controls are not descendants of `hP` this will not happen automatically and we have to make the necessary provisions ourselves. For plot `hP`, we define an application data field `usedByPslider` and store in it the handles of all `plotslider` controls. The `DeleteFcn` callback of `hP` is executed when `hP` is being deleted. We set the `DeleteFcn` property of `hP` to `@pDelFcn`, which has the following definition:

```
function pDelFcn(h,eD)
hGall=getappdata(h,'usedByPslider');
for k=1:length(hGall)
    D=getappdata(hGall(k),'LMData');
    delete(D.hUICM);
end
delete(hGall);
```

This function retrieves the handles `hGall` of the `plotslider` controls, deletes the `uicontextmenu` objects associated with each `hGall(k)` and then deletes all objects `hGall(k)`.

The code for the remaining callbacks is given below.

```
function bdwnFcn(hG,eD)
D=getappdata(hG,'LMData');
sType=get(D.hF,'SelectionType');
if strcmpi(sType,'normal')
    set(D.hF,'WindowButtonMotionFcn',{@moveFcn hG});
    set(D.hF,'WindowButtonUpFcn',{@stopFcn hG});
end

function moveFcn(h,eD,hG)
D=getappdata(hG,'LMData');
hA=D.hA;
```

```
hM=D.hM;
hL=D.hL;
xL=get(hA,'XLim');
yL=get(hA,'YLim');
currPt=get(hA,'CurrentPoint');
x=min(max(currPt(1,1),max(xL(1),D.X(1))),min(xL(2),D.X(end)));
set(hL,'XData',[x x]);
set(hM,'XData',x);
y=interp1(D.X,D.Y,x);
set(hM,'YData',y);

function stopFcn(h,eD,hG)
D=getappdata(hG,'LMData');
Xstr=num2str(get(D.hM,'XData'));
Ystr=num2str(get(D.hM,'YData'));
set(D.hUIM,'Label',['X=' Xstr ';' 'Y=' Ystr]);
set(h,'WindowButtonMotionFcn','');
set(h,'WindowButtonUpFcn','');

function delFcn(h,eD)
hG=get(h,'Parent');
D=getappdata(hG,'LMData');
delete(hG);
hGall=getappdata(D.hP,'usedByPslider');
vI=find(~(hGall==hG));
setappdata(D.hP,'usedByPslider',hGall(vI));
```

- The functionality of the control can be modified by changing the functions `moveFcn` and `stopFcn`.
- Each plot can have as many controls as we wish.
- Though the control in this example is designed to be modular, it still modifies the `DeleteFcn` of the plot `hP` and uses its application data.
- The control does not adjust itself when the axes change their `YLim` property. To address this problem we could consider defining the control in the hidden annotation axes.
- Examples of controls similar to `plotslider` are shown in Figure 7.

4.5 Predefined dialog boxes

MATLAB provides a set of predefined dialog boxes to be used for a spectrum of tasks, such as file input and output, printing, setting up font and colour of graphical objects. Enter `docsearch('Predefined Dialog Boxes')` to access the Helpdesk pages describing the functions which invoke them. Some of the standard dialog boxes, such as `inputdlg`,

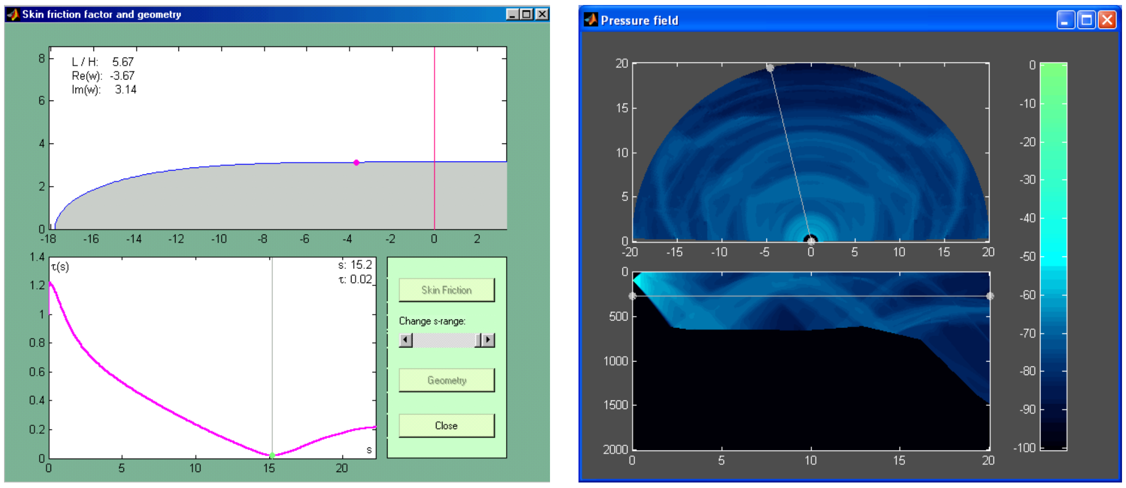


Figure 7: Controls similar to `plotslider`

`listdlg` or `waitbar`, use MATLAB figures, while some, e.g. `uigetfile`, `uiputfile` or `uigetdir` are not MATLAB-figure based.

There is also a standard MATLAB function `dialog`, which creates a figure with the properties recommended for dialog boxes. One of such properties is `WindowStyle`, and it is set by `dialog` to the string `'modal'`. If the resulting window is visible, it will trap all mouse and keyboard events occurring over all visible MATLAB windows which have been initialised earlier. Entering `Ctrl-C` resets the `WindowStyle` property of a modal figure to `normal`.

Exercise 28. In this exercise we consider the function `uigetfile`. The syntax for this command is

```
[fileN,pathN,filterIdx]=uigetfile(filterSpec,title)
```

where `fileN` is the selected file name, `pathN` is the file's path, and `filterIdx` is the selected filter index. Different filters are commonly specified using an $M \times 2$ cell array `filterSpec` so that elements `filterSpec{k,1}` (file extension to be shown) and `filterSpec{k,2}` (an associated string to display) describe filter number `k`.

The dialog box launched by `uigetfile` is no more than a tool for visual selection of files. After the files have been selected, their details are returned in the output variables which we can use in our subsequent file input and output tasks. The script in the following example (file `uigetf_tst.m` in the folder `Section4\dialogbx`) launches a `uigetfile` dialog box so that the user can select the `.png` or `.gif` files, upon a successful selection the `.jpg` versions of files are created in the folder `getf_out`. Note that the `Multiselect` option of the command is turned on.

```
fSpec={'*.png;*.gif','PNG or GIF files';...
    '*.png' 'PNG files'; '*.gif' 'GIF files'};
[fN,pN,fI]=uigetfile(fSpec,'Convert to JPG','Multiselect','on');
outDir='getf_out';
```

```
[s,m,mID]=mkdir(outDir);
if isequal(fN,0)
    return
elseif ~iscell(fN)
    fN={fN};
end
for k=1:length(fN)
    [pStr,fN0,fExt,v]=fileparts(fN{k});
    fExt=fExt(2:end);
    [D,map]=imread([pN filesep fN{k}],fExt);
    if ~isempty(map)
        imwrite(D,map,[outDir filesep fN0 '_' fExt '.jpg'],'jpg');
    else
        imwrite(D,[outDir filesep fN0 '_' fExt '.jpg'],'jpg');
    end
end
```

Exercise 29. This example considers a tool (see Figure 8) for simple manipulations with images. There are three different functions in the folder `Section4\crop`, `cropimg.m`, `imgcrop.m` and `imgcropm.m`, which have the same functionality (creating and driving virtually identical GUIs).

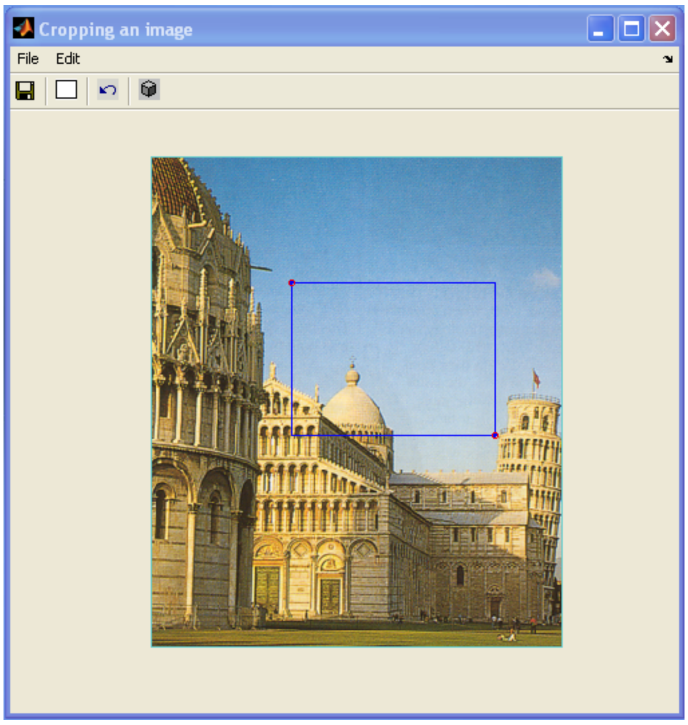


Figure 8: A crop tool

After running each of these functions examine their code (folder `Section4\crop`). In these notes we only provide the comments on the code.

- The functions `cropimg` and `imgcrop` use `feval` and callbacks specified as strings, whereas the callbacks in the function `imgcropm` are specified using function handles; `imgcrop` and `imgcropm` both have an additional `toolbar` with four `pushtools`, their callbacks are the same as those for the corresponding ui-menus.
- The following fields of the GUI figure's application data are used by the tool:

Application Data Field	*Data description*
`UsedByGUIData_m`	The `handles structure`
`ImageData`	Structure `ImgData` with a single field `CropImageMap` holding image colormap
`initXYLim`	Limits for the axes of the initial image
`PrevXYLim`	Axes limits at previous crop

Table 1: Application data fields used in Exercise 29

- Modify the code of any of the three functions so that defining the field `ImageData` in the GUI's application data can be avoided. Note that if the image is indexed, the function resets the `Colormap` property of the figure. How does the function retrieve the image matrix before saving its subset?
- The GUI uses dialogs `uigetfile` and `uiputfile` for its input and output tasks.
- Only axes limits are changed at each transformation, the `CData` property of the image remains the same during the entire cycle of manipulations with the loaded image.

5 Inclusion of ActiveX and Java Components

MATLAB's ability to incorporate ActiveX controls (for PC platforms) and Java components into its figure windows, and to interface them so that their properties and methods can be accessed programmatically is a significant addition to its versatility. Inclusion of ActiveX controls can be done using the function `actxcontrol`, while Java Swing components can be initialised and placed into a figure using `javacomponent`[10]. In this section we will experiment with these commands.

5.1 ActiveX controls

The MATLAB Helpdesk contains the background material for the Component Object Model (COM). We list below the titles of some useful articles about COM:

[10]There are also such higher-level commands as `uitable` and `uitree`.

- Programming with COM (to access press on the `Demo` tab, select Matlab, External Interfaces);
- COM Functions (use `docsearch('COM Functions')` to access the article);
- COM and DDE support (use `docsearch('COM and DDE support')` to access the article).

An ActiveX control can be created in figure `hf` using the command

```
hc=actxcontrol(progID,P,hf);
```

where `progID` is a programmatic identifier[11], `P` is the control's position in pixels in the format `[x0 y0 width height]`, `hc` is a handle to the control (try `disp(hc)` and/or `iscom(hc)`).

Creating an ActiveX control we intend to use its functionality in conjunction with the events it can trigger. To list all the events that can be fired by a COM object `hc` use `S=events(hc)`. An equivalent syntax is `S=hc.events`. The output `S` is a structure, its fields are event names and the contents of each field is a string describing the function prototype for the corresponding event's handler. The event will cause a handler function to execute only if it has been registered with this handler. To register an event use the function `registerevent`, as in the example below

```
f=@(h,eID,eD,eN) disp('f');
g=@(h,eID,eD,eN) disp('g');
C={'Click' f; 'DblClick' g};
h = actxcontrol('mwsamp.mwsampctrl.2',[0 0 200 200],gcf);
registerevent(h,C);
```

Registrering events in this example can also be done using `h.registerevent(C)`.

How did we know what arguments had to be used in the callback functions `f` and `g`? In order to work this out enter `S=h.events` and check `S.Click` and `S.DblClick` to find that the prototypes of event handlers must accept no arguments. Now, to obtain the required result, take into account that MATLAB passes the following arguments to the callback of a registered event:

Arg. 1: Object handle (format: MATLAB COM class);

Arg. 2: Event ID (format: double);

Next Args up to end-2: Arguments listed in event handler's prototype;

Arg. end-1: Event structure;

Arg. end: Event Name (format: character array).

[11]Provided by the vendor.

The event structure is summarized in the following table:

Field Name	Contents	Format
Type	Event Name	character array
Source	Control's handle	MATLAB COM class
EventID	Event Identifier	double
Event Arg Name 1	Event Arg Value 1	As in event handler's prototype
⋮	⋮	⋮
Event Arg Name N	Event Arg Value N	As in event handler's prototype

Table 2: Summary of the event structure

Event registration can also be done using `actxcontrol` at the stage of ActiveX control initialisation, the syntax for the case above would be

```
h=actxcontrol('mwsamp.mwsampctrl.2',[0 0 200 200],gcf,C);
```

where we recall that `C` was defined as `C={'Click' f; 'DblClick' g}`.

To obtain the list of all event/handler pairs that have been registered we can use either `S=eventlisteners(h)` or `S=h.eventlisteners`. We revoke registration of event/handler pairs using the function `uregisterevent`. For example, continue the above example by entering

```
unregisterevent(h,{'Click' f; 'DblClick' g});
disp(eventlisteners(h));
```

To find the details of ActiveX controls available on our computer system we can use the function `actxcontrollist`, which returns an $M \times 3$ cell-array, where M is the number of installed ActiveX controls. The elements of each row in this cell-array are the strings holding the name, the programmatic ID and the file name of the associated control. Another useful tool that can help us to obtain details about an ActiveX control is the GUI `activexcontrolselect`. If we call this function with two arguments, e.g. `[h,info]=actxcontrolselect`, then, upon the successful completion of the selection, we can obtain the three strings with the ActiveX control details from the 1×3 cell-array `info`. This function was designed to be used from within GUIDE.

Often ActiveX controls are objects with many parameters which can be changed. If we wish to save the state of a control to a file[12] we call the function `save`, if we wish to load the saved state from a file we use the function `load`[13].

Exercise 30. The file (`bsq.m`) for this exercise can be found in the folder `Section5`. We provide below an abbreviated version of the code.

```
function varargout=bsq(varargin)
```

[12]This procedure is also called serialization.

[13]Check help for `com/save` and `com/load`.

```
if nargin<=1
    FProp.Tag='bsqF';
       <some other figure properties: structure FProp>
    hf=figure(FProp);
    AxsProp.Tag='bsqA';
       <some other axes properties: structure AxsProp>
    ha=axes(AxsProp);
       <text properties: structure TProp>
    text(TProp);
    BProp.Style='pushbutton';
    BProp.String='Close';
    BProp.Tag='CloseB';
    BProp.Position=[100 20 70 20];
    BProp.Callback='bsq(''CloseBcbFcn'',gcbo,[],guidata(gcbo))';
    hB=uicontrol(BProp);
    hActX=actxcontrol('MSComctlLib.Slider.2',[30 55 140 20],hf);
    hActX.addproperty('Axes');
    hActX.Axes=ha;
    hActX.Min=0;
    hActX.Max=100;
    hActX.SmallChange=1;
    hActX.LargeChange=0;
    hActX.TickFrequency=10;
    hActX.registerevent({'MouseMove' @mMv; 'KeyDown' @kDF});
    handles=guihandles(hf);
    guidata(hf,handles);
    set(handles.bsqF,'HandleVisibility','off');
    if nargout==1
        varargout{1}=hf;
    end
elseif ischar(varargin{1})
    feval(varargin{:});
end
function CloseBcbFcn(h,eD,handles)
close(handles.bsqF);

function mMv(varargin)
hActX=varargin{1};
ha=hActX.Axes;
eD=varargin{end-1};
if eD.Button==1
    v=hActX.Value;
    set(ha,'Color',[v v v]/100);
end

function kDF(varargin)
hActX=varargin{1};
```

```
ha=hActX.Axes;
v=hActX.Value;
set(ha,'Color',[v v v]/100);
```

- This function initialises the simple GUI shown in Figure 9;
- Access from the callback functions of the ActiveX slider to the handle `ha` of the axes is provided by adding an additional property `Axes`;
- Definition of functions `@mMv` and `@kDF` implicitly takes into account the configuration of the input variables passed to the callback of the ActiveX by MATLAB.

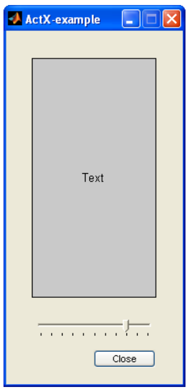


Figure 9: An ActiveX slider

The following exercise contains an analysis of how callbacks for ActiveX controls are dealt with in GUIDE-generated GUIs.

Exercise 31. Open GUIDE, select the `Blank GUI` template, press `OK`. Press on the `ActiveX Control` button in the Component Pallette, then use the mouse to select the area for the component. After the area rectangle has been selected, the `actxcontrolselect` GUI appears. Choose some control, for example Microsoft Slider Control version 6.0, then press `Create`. Now right-click on the ActiveX area, select `View Callbacks` and then, for example, `MouseMove`. At this stage MATLAB has to create an application m-file with the prototypes for the callback functions. When prompted, enter some name for the GUI, e.g. `mygui`, and press `Save`. An application m-file `mygui.m`, a figure file `mygui.fig` and a file `mygui_activex1` into which the state of the ActiveX control is serialized will be created as a result.

Now check that the name of the callback subfunction associated with the slider's `MouseMove` event is `activex1_MouseMove` and its prototype has the form

```
activex1_MouseMove(hObject, eventdata, handles) .
```
(1)

When GUIDE initialises an ActiveX control it creates an invisible peer uicontrol which holds the tag for the ActiveX, in our case this tag is `activex1`. For an ActiveX proxy, the corresponding field of the output structure returned by the function `guihandles` holds the handle to the associated COM object. If we enter

```
hf=mygui;
handles=guihandles(hf);
events(handles.activex1)
```

we will be able to see that the template of the `MouseMove` event handler is

```
MouseMove(int16 Button, int16 Shift, Variant x, Variant y)
```
(2)

In order to reconcile (1) with (2) let us check a few more things. The output of the command `eventlisteners(handles.activex1)` is a cell-array of two strings, `'MouseMove'` and `'mygui'`, which indicates that when the `MouseMove` event occurs over the ActiveX slider, MATLAB executes the function `mygui`, the main function of our application m-file. Since this is an event triggered by an ActiveX, MATLAB calls `mygui` with a certain set of input arguments (in our case, by (2), there will be eight arguments `a1` to `a8`), the first being the handle to the COM object, the second last being the event data structure, and the last argument being the event name.

By examining the initialisation code in `mygui.m` we can verify that it calls a function `gui_mainfcn` with the following input arguments:

`gui_State`, `hObj`, `a2`, `...`, `eD`, `'MouseMove'`,

where `hObj` is the COM object handle, `eD` is the event data structure, and the field `gui_Name` of the structure `gui_State` holds the string `'mygui'`. Note that the first input argument of `gui_mainfcn` is compulsory.

Now check that under these input arguments `gui_mainfcn` has to execute the commands

```
vin{1} = gui_State.gui_Name;
vin{2} = [get(varargin{1}.Peer, 'Tag'), '_', varargin{end}];
vin{3} = varargin{1};
vin{4} = varargin{end-1};
vin{5} = guidata(varargin{1}.Peer);
feval(vin{:});
return;
```

which in our case is equivalent to

```
mygui('activex1_MouseMove',hObj,eD,handles);
return;
```

The remaining part of the callback dispatch for the ActiveX of this example is the same as a callback invocation cycle for a standard object from the MATLAB graphics hierarchy.

5.2 Java components

Details on how to use Java from MATLAB can be found in MATLAB Helpdesk, e.g. use `docsearch('Calling Java from MATLAB')`, or in the MATLAB printable manuals [2, 3]. For our exercises, we just have to know that MATLAB provides an interface to Java which allows us to create objects from Java classes and access their properties and methods.

In the following example we experiment with the function `javacomponent`[14].

Exercise 32. The file `bsqj.m` can be found in the folder `Section5`. Its abbreviated code is provided below.

```
function varargout=bsqj(varargin)
if nargin<=1
    FProp.Tag='bsqF';
      <Other figure properties: structure FProp>
```

[14]This function introduced in R14 was revised in the Service Pack 1 of the release.

```
        hf=figure(FProp);
        AxsProp.Tag='bsqJ';
          <Other axes properties: structure AxsProp>
        ha=axes(AxsProp);
          <Text properties: structure TProp>
        hT=text(TProp);
        BProp.Style='pushbutton';
        BProp.String='Close';
        BProp.Callback='bsqj(''CloseBcbFcn'',gcbo,[],guidata(gcbo))';
          <Other push button properties: structure BProp>
        hB=uicontrol(BProp);
        s=javax.swing.JSlider;
        [s1,hc]=javacomponent(s);
        set(s,'Orientation',1,'UserData',ha,'Background',...
            [0.93 0.91 0.84],'DoubleBuffered','on','Value',0);
        set(hc,'Position',[130 20 20 100]);
        set(s,'StateChangedCallback',@chStateCb);
        handles=guihandles(hf);
        guidata(hf,handles);
        set(handles.bsqF,'HandleVisibility','off');
        set([hf, ha, hc, hB],'Units','normalized');
        if nargout==1
            varargout{1}=hf;
        end
    elseif ischar(varargin{1})
        feval(varargin{:});
    end

    function CloseBcbFcn(h,eD,handles)
    close(handles.bsqF);

    function chStateCb(varargin)
    h=varargin{1};
    ha=get(h,'UserData');
    v=get(h,'Value');
    set(ha,'Color',[v v v]/100);
```

The GUI resembles the examples considered for a MATLAB slider UI control and an ActiveX slider control (see Figure 10).
For events triggered by Java components, MATLAB uses the same pattern of input arguments passed to a callback as we had in the case of MATLAB standard graphics objects.

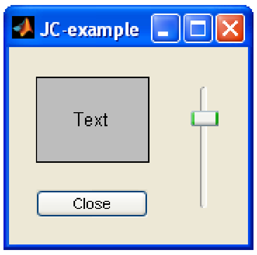


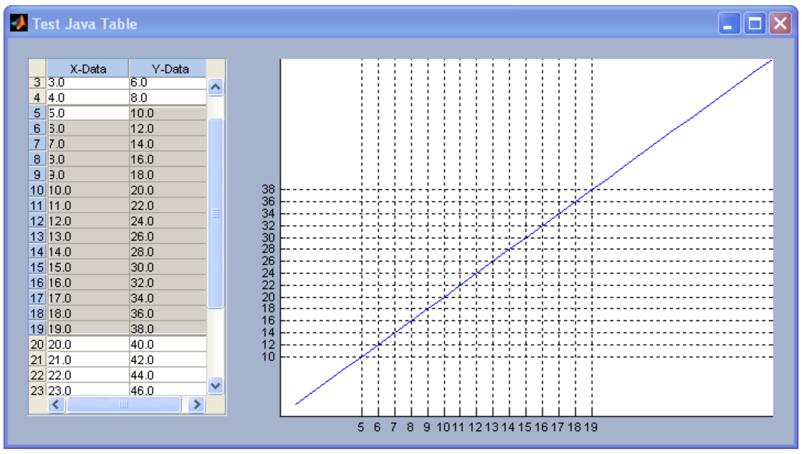
Figure 10: Slider Java cmponent

For example, when the callback is specified as a function handle, MATLAB expects that the corresponding callback function accepts at least two arguments, the handle to the source of the event, and the event data. As an experiment, replace the line containing `v=get(h,'Value')` in the body of `chStateCb` with

```
eD=varargin{2}; v=eD.getSource.getValue;
```

and check that the result gives the same functionality.

Exercise 33. This example tests the command `uitable`. Here we use MATLAB axes to visualise table selections.



	X-Data	Y-Data
3	3.0	6.0
4	4.0	8.0
5	5.0	10.0
6	6.0	12.0
7	7.0	14.0
8	8.0	16.0
9	9.0	18.0
10	10.0	20.0
11	11.0	22.0
12	12.0	24.0
13	13.0	26.0
14	14.0	28.0
15	15.0	30.0
16	16.0	32.0
17	17.0	34.0
18	18.0	36.0
19	19.0	38.0
20	20.0	40.0
21	21.0	42.0
22	22.0	44.0
23	23.0	46.0

Figure 11: MATLAB axes and a uitable object

The code for the function `tbj` of this example can be found in the file `tbj.m` in folder `Section5`. We had to make a few guesses in this example. What a method of the class does can often be worked out from the name of the method. A method's input arguments can be found using the command `methods` with the option `-full`, a visual representation of the output of this command can be obtained using the function `methodsview`.

6 Concluding Remarks

The aim of our exercises and code analysis was not to cover all aspects of GUI building in MATLAB, but rather to give a concise exposition of some possible viable GUI design techniques. Many useful guidelines for designing and programming MATLAB GUIs not considered here can be found in [1] or in the Helpdesk. For example, check the article about *Interrupting Executing Callbacks*[15], or examine details about various aspects of using GUIDE. Though we considered here relatively lightweight examples, their code can be used as a starting template for developing more complex GUIs.

Sometimes a GUI developed in MATLAB must be used on a computer which does not have MATLAB installed on it. The MATLAB Compiler allows developers to create stand-alone executables for their GUIs[16]. Such executables require a set of shared libraries called

[15]Code `intcbk.m` in folder `Section6` is provided for illustration.

[16]COM objects or Excel add-ins can be created using MATLAB Builder for COM or MATLAB Builder for Excel, these products can be used in conjunction with MATLAB Compiler.

the MATLAB Component Runtime (MCR). Application deployment procedures used in the previous release of MATLAB were different and used MATLAB C/C++ Math and Graphics Libraries. Consult [4] for more details.

References

1. The MathWorks Inc.: *MATLAB Creating Graphical User Interfaces*, MATLAB printable (pdf) documentation, Revision: 2004.

2. The MathWorks Inc.: *MATLAB External Interfaces*, MATLAB printable (pdf) documentation, Revision: 2004.

3. The MathWorks Inc.: *MATLAB External Interfaces Reference*, MATLAB printable (pdf) documentation, Revision: 2004.

4. The MathWorks Inc.: *MATLAB Compiler User's Guide*, MATLAB printable documentation, Revision: 2004.

Appendix A Comments on the course exercises

The exercises illustrate the following techniques and concepts:

[1–2] Handles of graphics objects, use of `get` and `set`; simple manipulation with object properties (here property `Parent` is examined).

[3] Examine `HandleVisibility` property of a figure; as an additional exercise, create a figure with `HandleVisibility` set to `Callback`:

```
hf=figure('HandleVisibility','Callback','ButtonDownFcn','disp(gcf-hf)');
```

(a) First try `hf1=gcf`, a new figure will be created ($\texttt{hf1} \neq \texttt{hf}$) (if `HandleVisibility` was set to `on`, the new figure would not be created and we would have that `hf1` would coincide with `hf`).

(b) Now click on figure `hf`, the callback executed in the workspace will display 0, i.e. `gcf` is `hf` ($\Rightarrow$ the figure `hf` is *visible* for `gcf` if it was called from *within* a callback).

(c) Now set the `HandleVisibility` property of `hf` to `off`

```
set(hf,'HandleVisibility','off');
```

and click on `hf` again, a new figure will be created ($\Rightarrow$ the figure `hf` is no longer visible to `gcf` even when it is called from within the callback).

[4] More on `HandleVisibility` of objects.

[5] Introductory example of a callback specified as a function handle; anonymous function (MATLAB7 feature) is used to define a function without creating an m-file.

[6] Verify that in the `(h,eD)` syntax format used by MATLAB, `h` is `gcbo`.

[7] This is an example of a callback (`KeyPressFcn`) which receives a non-empty event data.

[8–9] Examples with object sets which have different forms of hierarchies (also: conventions about parenting).

[10] Standard axes vs. annotation axes. In MATLAB there are a number of different classes of objects which have the same value of their `Type` property. One technique to check if the objects belong to different classes is to examine if they have the same sets of properties (define and use a simple function, `comp_sc`).

[11] Another example of different object classes with the same type (`line`).

[12] Different objects whose `Type` is `hgtransform` (Note: `hgtransform` is a R14 feature).

[13] UI controls and their callbacks.

[14] The Slider example using a MATLAB slider UI control.

[15] Obtaining handles of hidden objects (`uipushtool` and `uitoggletool` objects in a standard figure).

[16] UI menus and context-menus.

[17] MATLAB7 feature: `uibuttongroup`.

[18] MATLAB7 feature: `uipanel`; an example of a typical use of the cycle exploiting the properties `ButtonDownFcn`, `WindowButtonMotionFcn` and `WindowButtonUpFcn`.

[19] Test the constructs and commands: `varargin`, `varargout` and `feval`.

[20] Colon-indexed cell arrays as input and output arguments.

[21] Accessing subfunctions of a main function.

[22] `UserData`: why it is not the best choice for data exchange and storage. Application Data.

[23] `guidata` and `guihandles`.

[24] The Slider example using GUIDE.

[25] The Slider example using the code which mimics application m-file and `gui_mainfcn`.

[26] (a) `pd`-example: switchyard technique;
(b) `pd`-example: callback dispatch using `feval`;
(c) `pd`-example: specifying callbacks using function handle syntax.

[27] Example of a purpose-built control which uses descendants of axes (`plotslider`).

[28] Predefined dialogs: `uigetfile`.

[29] Predefined dialogs: 3 examples with the same functionality (image crop and conversion tool) which use different coding approaches.

[30] The Slider example using an ActiveX control.

[31] Analysis of callback dispatch for an ActiveX control in GUIDE generated application m-file.

[32] The Slider example using a Java component.

[33] Java components: a `uitable` example.

DISTRIBUTION LIST

Introduction to MATLAB Graphical User Interfaces

Sergey Simakov

AUSTRALIA

	Number of Copies
DEFENCE ORGANISATION	
Task Sponsor	
MOD/DSTO	1 (printed & CD)
S&T Program	
Chief Defence Scientist	1
Deputy Chief Defence Scientist Policy	1
AS Science Corporate Management	1
Director General Science Policy Development	1
Counsellor Defence Science, London	Doc Data Sheet
Counsellor Defence Science, Washington	Doc Data Sheet
Scientific Adviser to MRDC, Thailand	Doc Data Sheet
Scientific Adviser Joint	Doc Data Sheet and Dist List
Navy Scientific Adviser	1
Scientific Adviser Army	Doc Data Sheet and Dist List
Air Force Scientific Adviser	Doc Data Sheet and Dist List
Scientific Adviser to the DMO	Doc Data Sheet and Dist List
Systems Sciences Laboratory	
Chief of Maritime Operations Division	Doc Data Sheet and Dist List
Research Leader, MCS	Doc Data Sheet and Dist List
Research Leader, MSS	Doc Data Sheet and Dist List
Research Leader, LW	Doc Data Sheet and Dist List
Research Leader, MOR	Doc Data Sheet and Dist List
Head, UA Group	1 (printed & CD)
Head, ASW Sonar Systems Group	1 (printed & CD)
Head, Torpedo Systems Group	1 (printed & CD)
Zhi Yong Zhang	1

Andrew Knight	1
Garry Brown	1
Ashley Larsson	1
Colin Andrew	1
Jarrad Exelby	1
Paul Clarke	1
Graham Day	1
Author: Sergey Simakov	9 (printed & CD)
DSTO Library and Archives	
Library Edinburgh	2 (pdf, printed & CD)
Defence Archives	1
Capability Development Group	
Director General Maritime Development	Doc Data Sheet
Director General Land Development	1
Director General Capability and Plans	Doc Data Sheet
Assistant Secretary Investment Analysis	Doc Data Sheet
Director Capability Plans and Programming	Doc Data Sheet
Director General Australian Defence Simulation Office	Doc Data Sheet
Chief Information Officer Group	
Director General Australian Defence Simulation Office	Doc Data Sheet
AS Information Strategy and Futures	Doc Data Sheet
Director General Information Services	Doc Data Sheet
Strategy Group	
Director General Military Strategy	Doc Data Sheet
Assistant Secretary Governance and Counter-Proliferation	Doc Data Sheet
Navy	
Maritime Operational Analysis Centre, Building 89/90 Garden Island Sydney NSW Deputy Director (Operations) Deputy Director (Analysis)	Doc Data Sheet and Dist List
Director General Navy Capability, Performance and Plans, Navy Headquarters	Doc Data Sheet
Director General Navy Strategic Policy and Futures, Navy Headquarters	Doc Data Sheet
Air Force	
SO (Science) - Headquarters Air Combat Group, RAAF Base, Williamtown NSW 2314	Doc Data Sheet and Exec Summ

Army

ABCA National Standardisation Officer, Land Warfare Development Sector, Puckapunyal	e-mailed Doc Data Sheet
SO (Science) - Land Headquarters (LHQ), Victoria Barracks NSW	Doc Data Sheet and Exec Summ
SO (Science), Deployable Joint Force Headquarters (DJFHQ) (L), Enoggera QLD	Doc Data Sheet

Joint Operations Command

Director General Joint Operations	Doc Data Sheet
Chief of Staff Headquarters Joint Operations Command	Doc Data Sheet
Commandant ADF Warfare Centre	Doc Data Sheet
Director General Strategic Logistics	Doc Data Sheet
COS Australian Defence College	Doc Data Sheet

Intelligence and Security Group

AS Concepts, Capability and Resources	1
DGSTA , DIO	1 (printed & CD)
Manager, Information Centre, Defence Intelligence Organisation	1

Defence Materiel Organisation

Deputy CEO	Doc Data Sheet
Head Aerospace Systems Division	Doc Data Sheet
Head Maritime Systems Division	Doc Data Sheet
Program Manager Air Warfare Destroyer	Doc Data Sheet

OTHER ORGANISATIONS

National Library of Australia	1
NASA (Canberra)	1

UNIVERSITIES AND COLLEGES

Australian Defence Force Academy

Library	1
Head of Aerospace and Mechanical Engineering	1
Hargrave Library, Monash University	Doc Data Sheet

OUTSIDE AUSTRALIA

INTERNATIONAL DEFENCE INFORMATION CENTRES

US Defense Technical Information Center	1

UK Dstl Knowledge Services	1
Canada Defence Research Directorate R & D Knowledge & Information Management (DRDKIM)	1
NZ Defence Information Centre	1

ABSTRACTING AND INFORMATION ORGANISATIONS

Library, Chemical Abstracts Reference Service	1
Engineering Societies Library, US	1
Materials Information, Cambridge Scientific Abstracts, US	1
Documents Librarian, The Center for Research Libraries, US	1
SPARES	5 (printed & CD)

Total number of copies:	**Printed:**	**20**
	PDF:	**30**

Page classification: UNCLASSIFIED

DEFENCE SCIENCE AND TECHNOLOGY ORGANISATION DOCUMENT CONTROL DATA	1. CAVEAT/PRIVACY MARKING

2. TITLE Introduction to MATLAB Graphical User Interfaces	3. SECURITY CLASSIFICATION Document (U) Title (U) Abstract (U)
4. AUTHORS Sergey Simakov	5. CORPORATE AUTHOR DSTO Defence Science and Technology Organisation PO Box 1500 Edinburgh, South Australia, Australia 5111

6a. DSTO NUMBER DSTO–GD–0442	6b. AR NUMBER 013-447	6c. TYPE OF REPORT General Document	7. DOCUMENT DATE October, 2005

8. FILE NUMBER 2005/1055372/1	9. TASK NUMBER MOD/MGT	10. SPONSOR MOD/DSTO	11. No OF PAGES 47	12. No OF REFS 4

13. URL OF ELECTRONIC VERSION http://www.dsto.defence.gov.au/corporate/reports/DSTO–GD–0442.pdf	14. RELEASE AUTHORITY Chief, Maritime Operations Division

15. SECONDARY RELEASE STATEMENT OF THIS DOCUMENT

Approved For Public Release

OVERSEAS ENQUIRIES OUTSIDE STATED LIMITATIONS SHOULD BE REFERRED THROUGH DOCUMENT EXCHANGE, PO BOX 1500, EDINBURGH, SOUTH AUSTRALIA 5111

16. DELIBERATE ANNOUNCEMENT

No limitations

17. CITATION IN OTHER DOCUMENTS

No limitations

18. DEFTEST DESCRIPTORS

MATLAB, Graphical User Interfaces, Java components, ActiveX controls

19. ABSTRACT

The aim of this course is to develop initial skills for building Graphical User Interfaces (GUIs) in MATLAB7. First we give a summary of MATLAB's graphics object hierarchy and review the syntax for accessing and manipulating object properties. Then we discuss standard user interface components and consider situations when descendants of axes can be used to design purpose-built graphical controls. Programming techniques are analysed using moderately simple conceptual examples and exercises. The structure of application m-files generated by the MATLAB GUI development environment and some techniques for inclusion of Java components and ActiveX controls into MATLAB GUIs are also discussed.

Page classification: UNCLASSIFIED